AdventumRL: A Quest-Based Reinforcement Learning API

Kush Singh

Faculty Member 1: DocuSigned by: Mark Riedl D69235FF00B2469...

Faculty Member 2: DocuSigned by: 3CAC9A6AEF4F49F...

Table of Contents

Abstract

We propose AdventumRL, a framework to facilitate complex, quest-based reinforcement learning in Minecraft, a 3D first person sandbox video game. We define a set of encodings on top of tools provided by Malmo, an open-source Minecraft reinforcement learning framework. We define a grammar framework for encodings states, actions, transitions, and goals that can represent quests in a reinforcement learning scenario. Proof of concept reinforcement learning agents are provided: a tabular Q-learning agent, utilizing a table to determine the best action to take from a given state, and two different deep Q-learning agents, which utilize a neural network instead of a table to determine the best action to take from a given state. One of the deep Q-learning agents solely utilizes the camera feed from Minecraft to determine its location, while the other directly uses positional and coordinate information instead. We demonstrate that the addition of our grammar framework allows the agents to complete a locked room quest that they could not complete without it.

Introduction

Reinforcement learning is the process through which an agent learns how to act in certain scenarios in order to maximize the expected future reward. While the agent receives rewards after actions it takes, the agent does not know how to can gain positive rewards and must discover so through trial and error. Furthermore, the agent must determine which of the actions in the series of actions that were taken was ultimately responsible for granting the rewards: the 'credit assignment' problem [16].

Since its development in the late 1990's, reinforcement learning has become an increasingly popular area in the field of machine learning, especially since the advent of deep reinforcement learning. Deep reinforcement learning, reinforcement learning approaches that take advantage of neural networks, has enabled agents to tackle more challenging problems [9]. Neural networks consist of a series of connected “artificial neurons,” or nodes. These nodes take in signals, apply a function, and propagate the signal to other nodes if certain thresholds for the signal are met. Thus, a large amount of information can be utilized as an input signal, which is processed by the neural network and results in an output signal that directs how an agent should act. One particularly popular field of exploration for reinforcement learning has been training agents to play video games, starting from the simplistic games of the Atari 2600 [10, 12] and now advancing into more complicated games, such as Defense of the Ancients 2 (Dota 2) [11] and StarCraft 2 [13, 17].

Another video game that has become a popular target for reinforcement learning research is Minecraft. Minecraft is a 3D first person "sandbox" game where the player is free to explore infinitely large procedurally generated worlds, building and interacting with various entities as he or she wishes. Due to the openness of the game, a large variety of tasks and challenges can be designed for a potential agent to tackle. Indeed, several researchers have utilized Minecraft as a "test bed" for novel deep reinforcement learning architectures and techniques [3, 4, 5, 6]. Commonly, reinforcement learning research in Minecraft makes use of Project Malmo, the state-of-the-art platform for reinforcement learning in Minecraft. Malmo serves as the software bridge between reinforcement learning agents and Minecraft, making it easy for agents to receive information from and send commands back into the game [7].

While a significant amount of work has gone into developing Malmo, Malmo lacks the capability to support more complex, mission-based quests. We define quests as a series of discrete high-level goals that must be achieved to fulfill an objective, such as finding and utilizing a set of keys to unlock a series of doors and ultimately escape a dungeon. A goal is a particular configuration of facts, propositions about the world, that the agent must fulfill through taking actions. Quest-based reinforcement learning is a reinforcement learning scenario that structures the task to be learned as a quest. As opposed to general reinforcement learning, where there is no intrinsic structure to separate higher level facts and transitions about the world from the general state space, quest-based learning can require several low-level actions to be taken in order for a single transition in the propositional space. Malmo encodes a map, reward signal, observations, and actions through a MissionSpec [7], but still struggles to communicate high-level objectives that are not just navigation tasks.

In our research, we propose AdventumRL, a novel framework that extends Malmo with a grammar to encode quest states, goals, actions, and transitions as high-level abstractions. By allowing developers to simplify action spaces and track high-level information while serving as an intrinsic method to describe complex propositional structures, AdventumRL facilitates the development of richer reinforcement learning tasks in Minecraft. One possible use could be to specify rewards for the completion of subtasks, allowing for the completion of harder exploration tasks, like the key and door example mentioned earlier. AdventumRL defines a set of encodings on top of Project Malmo where states are represented through first-order logic on problem space entities and several methods serve to create a logical grammar schema. In order to track transitions in the overarching high-level facts and propositions, individual low-level actions taken by the agent will be tracked to determine when preconditions are met and what postconditions are triggered, inspired by TextWorld [8]. AdventumRL also includes sample agents that can solve simple quests that agents using just Malmo struggle with.

As a result of our research, we facilitate novel explorations in reinforcement learning through quests by contributing:

1. AdventumRL, a novel framework to support quest-based reinforcement learning in Minecraft

2. An outline of applications for AdventumRL, alongside a structure to learn high-level concepts

3. A series of proof-of-concept agents and quests, as well as an interface for building new agents and quests in AdventumRL

Literature Review

With the advent of deep reinforcement learning, agents have become capable of tackling more challenging problems, especially ones that heavily involve visuals. Deep reinforcement learning makes use of multi-layered neural networks, which are extremely efficient at finding numerical representations of images, such as frames of a video game. As a result, deep reinforcement learning allows agents to quickly understand and interact with video games. Notably, O. Vinyals et al. have utilized deep multi-agent reinforcement learning to create an agent that can play StarCraft 2 without any simplifications or other "crutches" and achieve a competitive ranking better than 99.8% of human players [13]. Similarly, OpenAI, et al created a large-scale deep reinforcement learning AI system that was able to defeat the world champions in Dota 2, a world first for AI in e-sports [11].

We now examine several different papers that showcase reinforcement learning innovations in Minecraft. First, Oh et al. outline a series of pathfinding and recognition tasks in Minecraft and go on to present a novel deep reinforcement architecture that has superior performance on these tasks compared to conventional methods [3]. It is interesting to note how their previous work included the creation of deep neural networks to predict future frames in Atari arcade games like Pac-Man [12], because this shift highlights how Minecraft is a becoming a popular focus in current reinforcement learning research. Next, Alaniz defines a Minecraft block placing task, on which a novel framework's performance is shown to be more effective than alternatives [4]. Frazier and Riedl introduce the concept of action advice agents to support a deep reinforcement learning agent to overcome perceptual aliasing, the phenomenon where several different in-game states appear similar visually, making it difficult for an agent to appropriately determine which state it is currently occupying. An agent assisted with different types of action advice is then given a navigation task designed to induce perceptual aliasing [5]. Finally, Matsui et al. present experiments into how differences in an agent's angle of sight in Minecraft can affect its performance in a pathfinding task [6].

Despite the innovations being presented or challenges addressed, we observe that the agents were all tested on relatively simple navigation and building tasks in Minecraft. These tasks are typically simple actions for human players, who instead utilize them as the building blocks through which more complex goals can be achieved. With AdventumRL, it would be

possible for novel agents to easier explore more difficult reinforcement learning tasks, such as hard exploration scenarios like Montezuma's Revenge [15]. Montezuma's Revenge is a particularly difficult exploration problem that features sparse rewards, where the lack of reward given to the agent further exacerbates the credit assignment problem.

In summary, we propose AdventumRL, a framework for quest-based reinforcement learning in Minecraft, through a series of encodings that will be built on top of Project Malmo. Specifically, we will provide a grammar to encode key quest components, such as states, actions, transitions, and goals, allowing for the high-level constructs of a quest to be easily tracked and evaluated. As a proof of concept, we include a tabular Q-learning agent, and two deep Q-learning agents. Q-learning is an approach to reinforcement learning where the agent attempts to learn the function $Q(s, a)$, an estimate of the future reward achieved for taking an action a from the current state s, through trial and error. A tabular-Q agent models the future reward for performing a particular action from a state as a discrete table mapping actions from every state to expected rewards, whereas a deep Q-learning agent utilizes a neural network to estimate the future reward for an action. We believe that AdventumRL will allow for the exploration of novel reinforcement learning questing scenarios in Minecraft, opening the door for interesting research and potentially serving as a stepping-stone for the training of agents to learn more complicated tasks in the real world.

Methodology

The methodology for AdventumRL comprises of three major sections: the quest grammar, the agent models, and testing. The grammar consists of the logic and mechanisms that allow agents to understand the conditions of a quest and work towards solving them. Next, the agent models consist of three different approaches to solving the same reinforcement learning challenge. Finally, there is the testing process, through which agents are evaluated to determine if the addition of grammar provides a significant advantage in learning questing scenarios.

Quest Grammar

AdventumRL's grammar creates a propositional space, a high-level representation consisting of simple state information, such as facts and propositions about the world. Facts are presently true relations between two elements, whereas propositions are presently true relations between two sets of elements. Our knowledge representation approach is inspired by first-order logic and the work done by TextWorld [8]. To perform a transition between states in the propositional space, actions must be taken. Each action has preconditions that need to be satisfied before the action can be performed, and result in a set of postconditions becoming true. States that can be reached through a combination of these actions can also be designated as subgoals for the agent to work towards.

The grammar's propositional space can hold information about objects, relations, and functions. For example, in order to create a mission where the objective is to collect a gold ingot and a diamond, the initial state can be represented as $in(gold, house) \wedge in(diamond, chest)$; the gold is in the house and the diamond is in the chest The action to collect the diamond would be $in(diamond, chest) \wedge by(agent, chest) \rightarrow in(diamond, inventory)$; if the diamond is in the chest and the agent is by the chest, the agent can collect the diamond, resulting in the diamond being in the inventory of the agent. Goals are similarly represented as a set of facts and propositions in the space, such as $\forall\ ores | in(ores, inventory)$; every ore is in the agent's inventory.

The grammar contains several built-in predicates that can be used to define facts, propositions, actions, and goals like the ones in the aforementioned example. AdventumRL automatically computes the present values of these predicates, through routines inspired by [14].

- in: Checks to see if entity A's coordinates are contained by the coordinates of entity B or if an item is in an agent's inventory. For example, can describe if an agent is inside a specific building, or if the agent has collected a key item.
- at: Checks if entity A shares its coordinates with entity B
- by: Checks if entity A is within interaction range of entity B
- unlocked: Checks whether an unlockable item is currently unlocked
- inhand: Checks whether an entity is currently in the agent's first inventory slot
- hasMaterials: Checks whether the agent has the requisite materials in their inventory to craft an item

Users can specify constructs using these predicates by describing them in a grammar file. Additionally, Minecraft entities can be detailed in a quest file, which allows them to be tracked with respect to the predicates. As a result, specific objects like keys and doors can be given unique identifiers. Abstract bounding boxes can also be designated as locations, without any relation to physical structures inside Minecraft itself.

Triggers, facts or propositions whose values are a part of the general state space, can also be specified. They allow for additional information to be directly given to the agent alongside the more general information like the agent's current location. For example, a trigger could designate what types of blocks the agent is standing on, to aid with navigation through a maze.

AdventumRL also supports the ability to only pass visual information directly from Malmo to the agent, or any combination of Malmo information and grammar information as desired by a developer.

Agent Models

We created three premade agents to demonstrate the capabilities of AdventumRL; a tabular agent (TabQ agent), a Deep Q-Learning agent using propositional features (DQN agent), and a Deep Q-Learning agent using camera features (Camera DQN agent). While the TabQ agent employs a standard q-learning table to understand and navigate a discrete state-action space, the DQN agents make use of a generalized embedding to approximate and navigate a continuous state-action space. Specifically, the Camera DQN agent uses the actual game footage

to determine its current position instead of receiving direct coordinates and positional information from Malmo like the DQN agent does.

Q-learning is an approach to reinforcement learning where the agent attempts to learn the function $Q(s,a) = \alpha_t(R(s) + \gamma\, max_a Q(s_{t+1}, a))$, where the function output is determined by a learning rate multiplied to the reward for entering the current state, plus a discount factor (for reducing the value of future actions) multiplied by the maximum possible q-value that can be achieved by future states reachable from state s after taking action a. This equation is the Bellman equation for Q-learning, and is learned by the agent through trial and error. An agent can receive positive rewards for completing an intended task, or additionally for accomplishing steps along the way, but can also receive negative rewards for failing in an unrecoverable manner, such as dropping the player character into lava and dying. The agent attempts to learn the function in order to maximize the reward it receives upon every attempt of the quest, through choosing the actions that are estimated to lead to the greatest future reward.

Tabular Q-learning, the variety used by our first agent, represents the Bellman equation as a table of discrete states and discrete actions. As a result, our agent considers every space it could be standing on as a different state. Additionally, every state also includes the triggers as defined in the grammar. Since there are a total of *s* different spaces the agent can be occupying, and the triggers can take *t* different values depending on the agent's actions, there are a total of *s*t* total states for the agent to explore. From any given state, the agent can move forward, backwards, left, right, interact with items (for example, picking up or dropping an item), or perform actions in the propositional space, if the preconditions are met. Depending on how the quest is defined, the agent can receive intermediate rewards for performing actions or causing triggers to change.

Our second agent utilizes a Deep Q Network (DQN), which is a neural network that instead of discretely calculating the Q-value of every single entry in the table, approximates the Bellman equation, *Q(s, a).* The agent takes in information about the current position occupied in Minecraft and the current propositional space. The neural network interprets the triggers and action preconditions from the propositional space as their own one-hot embeddings, while also reading continuous information, like position, as integer embeddings. One-hot embeddings are binary variables that indicate the current state of the triggers and preconditions, with 0

corresponding to false and 1 corresponding to true. The network then utilizes four fully connected layers (31x31, 31x23, 23x15, and 15x8) with LeakyReLU activation functions. LeakyReLU is an activation function that outputs the input exactly if greater than zero and one percent of the input if it is less than zero. The architecture is summarized in *Figure 1.* Finally, the output is every possible action's corresponding Q-value, allowing for the selection of the action with the highest Q-value, which is expected to bring about the highest reward.

Our third agent also uses a DQN, but instead of receiving direct coordinates and positional information from Malmo about its location and surroundings, it instead receives the camera feed from Minecraft. To localize itself, the agent utilizes a Convolutional Neural Network (CNN) to extract image features from the camera feed. Convolutional neural networks are neural networks that are especially tailored to process images. CNN's treat each image as a collection of three two-dimensional matrices (one matrix for each primary color: red, blue, and green), with each entry in the matrix consisting of how much of that respective color is present in the part of the image corresponding to the entry. A CNN consists of convolution layers, pooling layers, and fully connected layers. In convolution layers, the input matrices are convolved with a kernel of a predetermined size (*n* x *n)*. Convolving with a kernel is a mathematical operation where every (*n* x *n)* cluster of the input matrix is element-wise multiplied to the kernel, and the result is summed to create a new corresponding entry in the smaller output matrix. The corresponding equation is $G[a, b] = \sum_x \sum_y k[x, y] i[a - x, b - y]$, where G[*a, b*] is the value of output matrix at coordinate *(a,b)*, *i* is the input image, *k* is the kernel, and *(x, y)* also represent coordinates of the input and kernel. Pooling layers can either be max pooling or average pooling. In max pooling layers, the largest value in every (*m* x *m*) cluster of the input matrix is chosen to create a corresponding entry in an output matrix, while in average pooling layers, the average of every (*m* x *m*) cluster is chosen for the output. Fully connected layers flatten the convolved and pooled output into a column and treat it as an input for a standard neural network to produce a final output.

The Camera DQN agent also approximates *Q(s, a)* for every possible action *a* that can be taken from state *s*. However, different from the previous two agents, *s* has *w*h*p* dimensions, where *w* and *h* are the width and height of the game screen respectively and *p* is the size of the proposition space consisting of the triggers and action preconditions from AdventumRL. Our

model utilizes two convolutional layers with a 172x96 frame input, 5x5 kernel sizes, 2x2 max pooling, and 6 and 16 output channels, respectively. Afterwards, there are two fully connected layers of 728x128 and 128x8. One final difference with the Camera DQN is its possible actions. Instead of moving forward, backwards, left, or right, the agent instead can only move forward or turn ninety degrees clockwise and counterclockwise. The alternative movement scheme allows the agent to better view and therefore understand the game world.

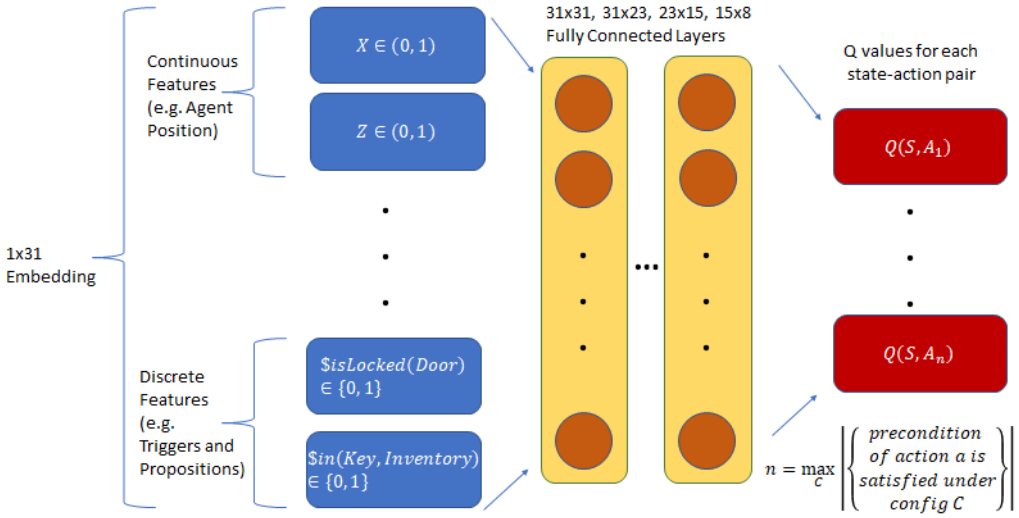

Figure 1: Architecture for DQN Model & Embedding

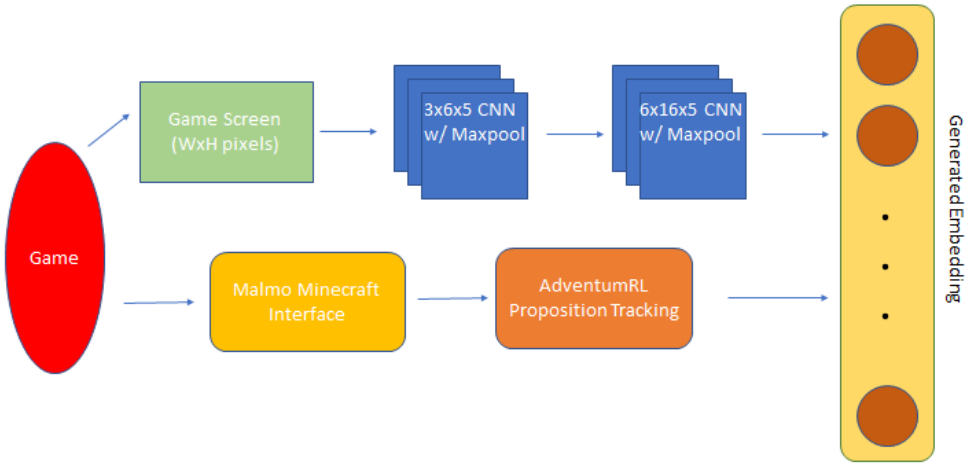

Figure 2: Architecture for Camera CNN Generated Embedding

Testing

To demonstrate AdventumRL and test the agents, we created a simple quest where the agent has to learn to first pick up a key and then use it to unlock a door in order to escape. At the same time, the agent needs to avoid failing through falling into the lava. As can be seen in *Figure 3*, the key is represented by a diamond, which needs to be thrown onto the gold block to unlock the door, which represented by the dark blue lapis lazuli block. The quest is designed to determine whether the grammar provided by AdventumRL allows for a significant advantage when training agents to handle challenges that require operations in a particular sequence (get the key, unlock the door, then exit) to succeed.

While picking up and dropping the key are handled by standard Minecraft mechanics, the unlocking of the door is defined in the grammar as: *locked(door) ∧ by(agent, door) ∧ in(key, inventory) ∧ inhand(key) → !locked(door) ∧ by(agent, door)*. In other words, if the door is locked and the agent is by the door with the key in the inventory and in their hand, the agent can unlock the door, using the key and resulting in the door being unlocked, with the agent still standing by the door. Furthermore, we included triggers for the agent having the key in the inventory and the current state of the door (locked or unlocked). The additional knowledge provided by the triggers aid training, since the agent can easier explore and understand the different state configurations it enters and exits while exploring the environment.

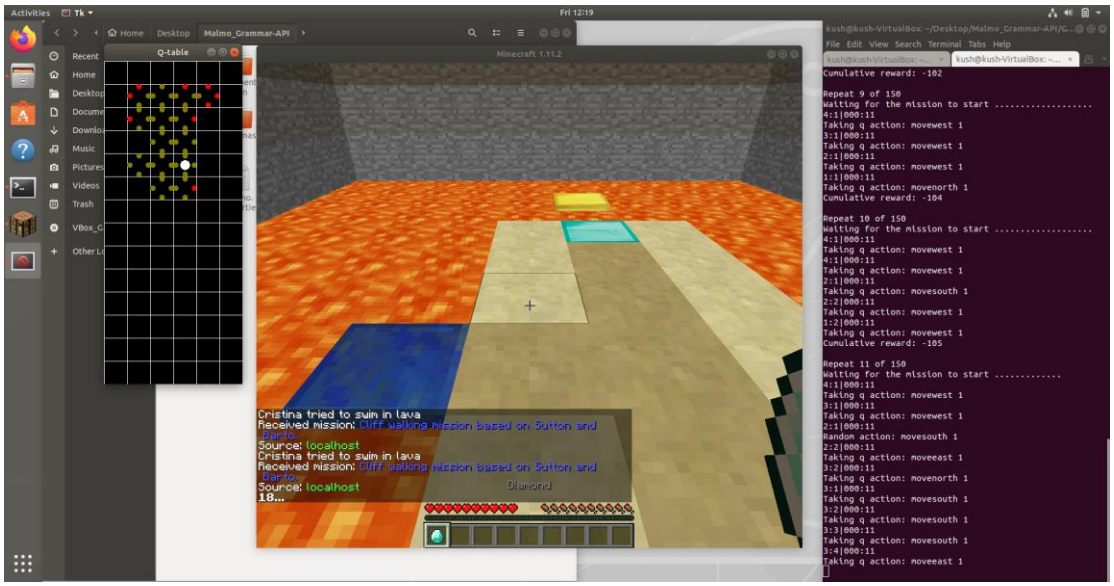

Figure 3: The TabQ Agent Attempting the Locked Room Quest

Discussion

As outlined in the methodology, we tested all three versions of our agents, TabQ, DQN, and Camera DQN, in the locked room quest with and without the AdventumRL grammar providing preposition and trigger information to the agents.

The TabQ agent without triggers and propositional information quickly got caught in a local minimum after just twenty-five iterations, as can be observed in *Figure 4*. Due to the fact that each state consists not just of the agent's current location in the world, but also information like status of the door and the current location of the key, the number of states the agent has to explore increases exponentially over time. However, the agent is unable to properly navigate these states. As a result, the agent occasionally succeeds in grabbing the key, but fails to proceed any further in the sequence to escape. Exploration problems are a common issue in reinforcement learning tasks, especially in quests, which was the original impetus for creating AdventumRL. In comparison, the TabQ agent that did receive grammar information was able to converge to the optimal policy within a hundred iterations. In *Figure 5*, the major upticks in cumulative reward at fifty and one hundred iterations denote points where the trigger state representation changed, denoting that the agent learned to pick up the key and unlock the door, respectively.

The DQN agent without triggers and propositional information faced the same difficulties that the TabQ agent without grammar did, in that the quest simply contains so many states that the agent is unable to properly explore and determine the best course of action. After about forty-five hundred iterations, the DQN agent stabilizes to a suboptimal policy, as seen in *Figure 6*. On the other hand, the DQN agent with triggers and propositions converges to a successful solution after thirty-five hundred iterations (*Figure 7)*. It is also important to note that the training process for DQN agents are generally more unstable than their tabular counterparts, hence the occasional drops in reward even after the solution was found.

The Camera DQN agent without triggers and propositions failed to find a winning policy, just like the regular DQN agent (*Figure 8)*. However, the Camera DQN agent with triggers and propositional information begins finding the key almost immediately, starts intermittently

unlocking the door within a few hundred iterations, and converges to a solution around thirteen hundred iterations (*Figure 9)*. The instability of DQN agents is visible, as the Camera DQN agent only manages to escape about half the time, even after finding the solution.

These results serve as the proof of our concept: that AdventumRL is a system that successfully allows reinforcement learning agents to solve complex quest-based scenarios. As a result, future research can utilize AdventumRL to create more robust reinforcement learning challenges for their experiments.

Some immediate next steps that we are currently exploring include are investigating the performance of all the agents, with and without grammar, on other quests.

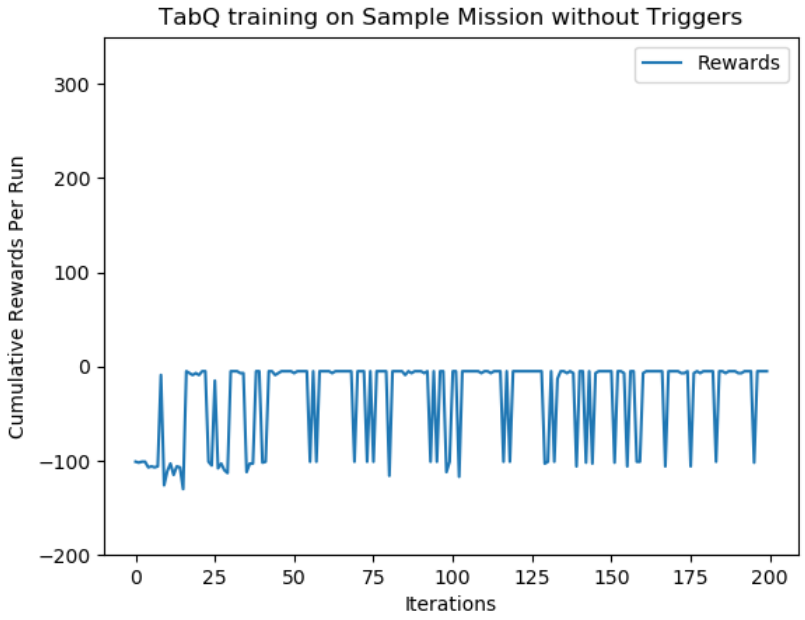

Figure 4: Cumulative rewards for TabQ agent in the locked room quest

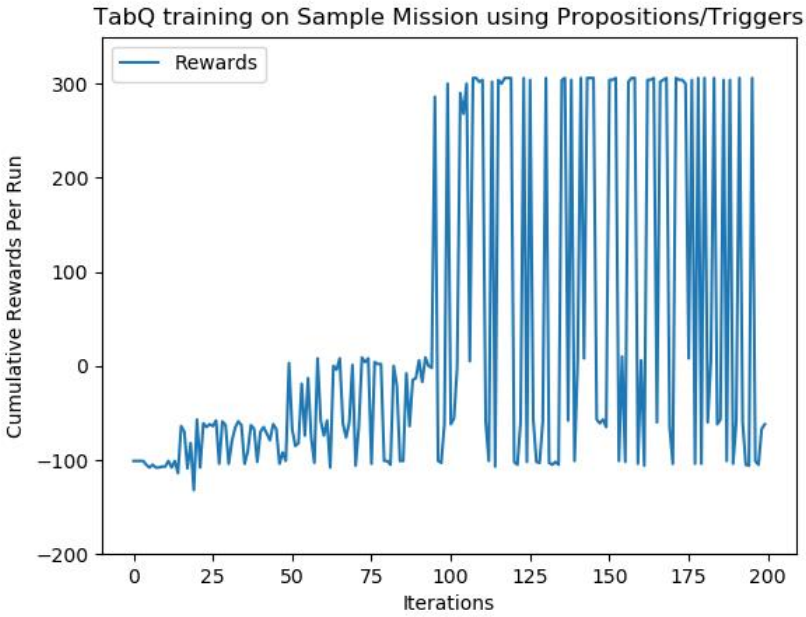

Figure 5: Cumulative rewards for TabQ agent in the locked room quest using AdventumRL triggers and propositional information

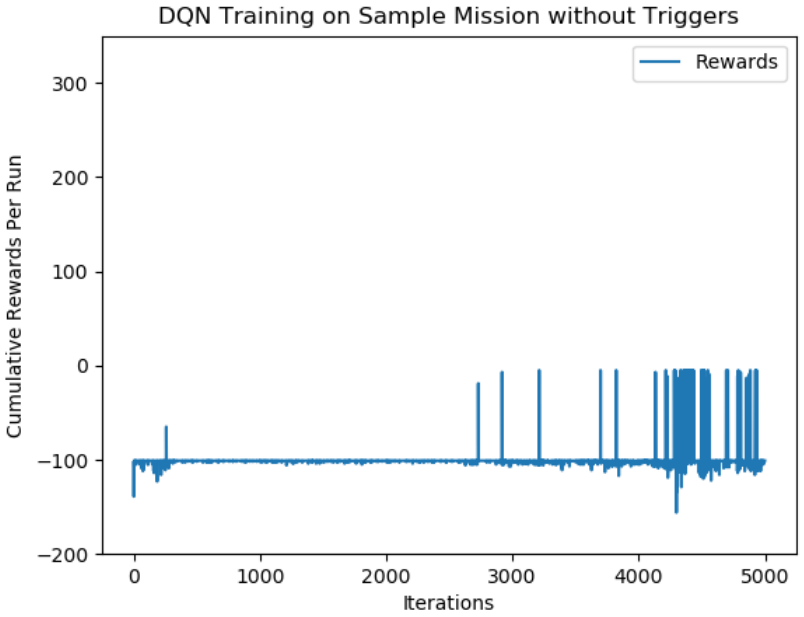

Figure 6: Cumulative rewards for DQN agent in the locked room quest

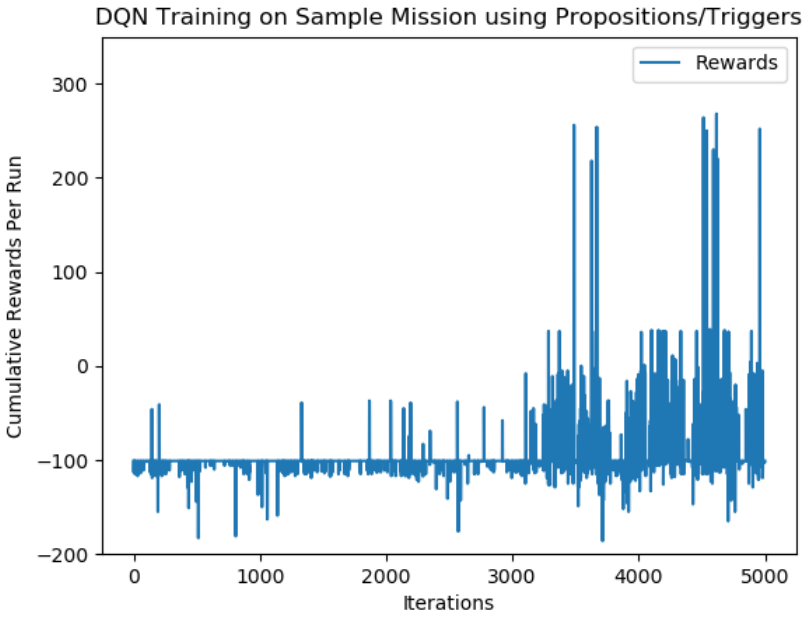

Figure 7: Cumulative rewards for DQN agent in the locked room quest using AdventumRL triggers and propositional information

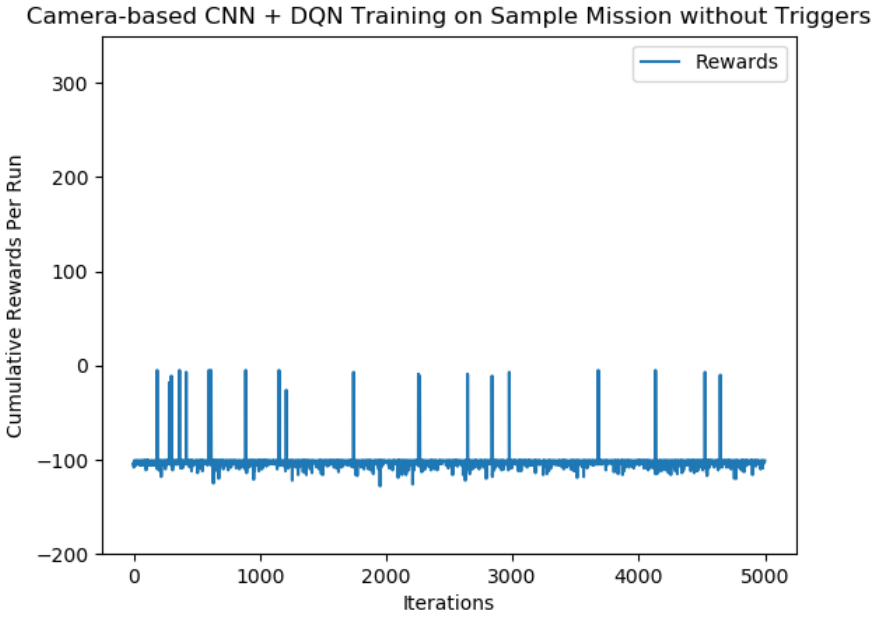

Figure 8: Cumulative rewards for Camera DQN agent in the locked room quest

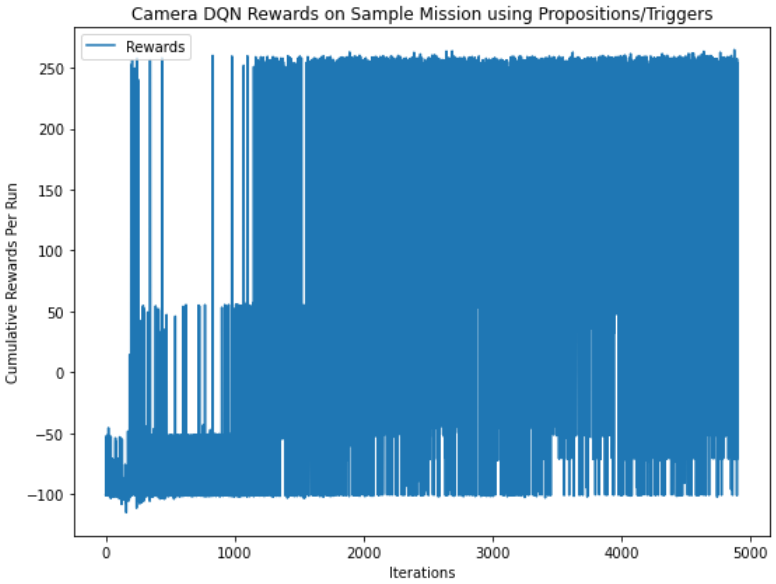

Figure 9: Cumulative rewards for Camera DQN agent in the locked room quest using AdventumRL triggers and propositional information

Conclusion

We described AdventumRL, a novel framework that extends Malmo with a grammar to encode quest states, goals, actions, and transitions as high-level abstractions, in order to facilitate quest-based reinforcement learning. We provided a set of value propositions and use cases for the framework and defined a structure for representing high-level constructs. Finally, we provided a set of proof-of-concept models as well as an interface for building new quests and agents.

All in all, we believe that AdventumRL will allow Minecraft to serve as a more versatile testing environment for reinforcement learning research. Instead of simply working with block placing or basic navigation tasks, agents will now be able to be pitted against a rich variety of complex quests. Some different future applications for AdventumRL could include reducing joint-action spaces, the number of possible configurations multiple agents can be occupying during a mission, for multi-agent reinforcement learning or tackling especially difficult exploration tasks, such as in the game Montezuma's Revenge. [15]

References

[1] M. Brittain and P. Wei, “Hierarchical Reinforcement Learning with Deep Nested Agents,” *arXiv:1805.07008 [cs]*, May 2018, Accessed: Sep. 20, 2020. [Online]. Available: http://arxiv.org/abs/1805.07008.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

[3] J. Oh, V. Chockalingam, S. Singh, and H. Lee, “Control of Memory, Active Perception, and Action in Minecraft,” *arXiv:1605.09128 [cs]*, May 2016, Accessed: Sep. 20, 2020. [Online]. Available: http://arxiv.org/abs/1605.09128.

[4] S. Alaniz, “Deep Reinforcement Learning with Model Learning and Monte Carlo Tree Search in Minecraft,” *arXiv:1803.08456 [cs, stat]*, Mar. 2018, Accessed: Sep. 20, 2020. [Online]. Available: http://arxiv.org/abs/1803.08456.

[5] S. Frazier and M. Riedl, “Improving Deep Reinforcement Learning in Minecraft with Action Advice,” *arXiv:1908.01007 [cs, stat]*, Aug. 2019, Accessed: Sep. 20, 2020. [Online]. Available: http://arxiv.org/abs/1908.01007.

[6] T. Matsui, S. Oyama, and M. Kurihara, “Effect of Viewing Directions on Deep Reinforcement Learning in 3D Virtual Environment Minecraft,” in *PRIMA 2018: Principles and Practice of Multi-Agent Systems*, vol. 11224, T. Miller, N. Oren, Y. Sakurai, I. Noda, B. T. R. Savarimuthu, and T. Cao Son, Eds. Cham: Springer International Publishing, 2018, pp. 527–534.

[7] Johnson, Matthew and Hofmann, Katja and Hutton, Tim and Bignell, David, *The Malmo Platform for Artificial Intelligence Experimentation*. AAAI Press.

[8] M.-A. Côté *et al.*, “TextWorld: A Learning Environment for Text-based Games,” *arXiv:1806.11532 [cs, stat]*, Nov. 2019, Accessed: Sep. 20, 2020. [Online]. Available: http://arxiv.org/abs/1806.11532.

[9] Arulkumaran, Kai, et al. “A Brief Survey of Deep Reinforcement Learning.” *IEEE Signal Processing Magazine*, vol. 34, no. 6, Nov. 2017, pp. 26–38. *arXiv.org*, doi:10.1109/MSP.2017.2743240.

[10] Mnih, Volodymyr, et al. “Playing Atari with Deep Reinforcement Learning.” *ArXiv:1312.5602 [Cs]*, Dec. 2013. *arXiv.org*, http://arxiv.org/abs/1312.5602.

[11] OpenAI, et al. “Dota 2 with Large Scale Deep Reinforcement Learning.” *ArXiv:1912.06680 [Cs, Stat]*, Dec. 2019. *arXiv.org*, http://arxiv.org/abs/1912.06680.

[12] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh, “Action-Conditional Video Prediction using Deep Networks in Atari Games,” *arXiv:1507.08750 [cs]*, Dec. 2015, Accessed: Nov. 19, 2020. [Online]. Available: http://arxiv.org/abs/1507.08750.

[13] O. Vinyals *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019, doi: 10.1038/s41586-019-1724-z.

[14] R. Michael Young, Mark O. Riedl, Mark Branly, Arnav Jhala, R.J. Martin, and C.J. Saretto, “An architecture for integrating plan-based behavior generation with interactive game environments,” *Journal of Game Development*, pp. 51–70, 2004.

[15] T. Salimans and R. Chen, “Learning Montezuma’s Revenge from a Single Demonstration,” *arXiv:1812.03381 [cs, stat]*, Dec. 2018, Accessed: Sep. 20, 2020. [Online]. Available: http://arxiv.org/abs/1812.03381.

[16] M. Minsky, “Steps toward Artificial Intelligence,” Proc. IRE, vol. 49, no. 1, pp. 8–30, Jan. 1961, doi: 10.1109/JRPROC.1961.287775.

[17] A. Silva and M. Gombolay, “Neural-encoding Human Experts’ Domain Knowledge to Warm Start Reinforcement Learning,” arXiv:1902.06007 [cs, stat], Sep. 2020, Accessed: May 06, 2021. [Online]. Available: http://arxiv.org/abs/1902.06007.